National Aeronautics and
Space Administration

Jet Propulsion Laboratory
California Institute of Technology
Pasadena, California

Is Systems Engineering Really Engineering?

Steven Jenkins
Principal Engineer
Systems Engineering and Formulation Division

Is Systems Engineering Really Engineering?

- The question is rhetorical
- Of course, what I mean is "How do we ensure that systems engineering really is engineering?"
- To answer that, we first have to know what characterizes engineering
- It's too big a job to define engineering, but we can talk about some necessary conditions

What Do Engineers Do?

- **Engineers do two complementary things:**
  - they *describe* actual and imagined states of the world
    - actual states are facts
    - imagined states are designs and consequences
  - they *analyze* these descriptions
    - What are the consequences of a specified design?
    - What designs have a specified set of consequences?
- **Engineering analysis is distinguished by its reliance on science and mathematics to achieve rigor**
- **What is rigor?**
  - the quality or state of being very exact, careful, or strict
    - Merriam-Webster, 2017
  - scrupulous adherence to established standards for conduct of work
    - NASA Final Report of the Return to Flight Task Group, Appendix A.2, 2005

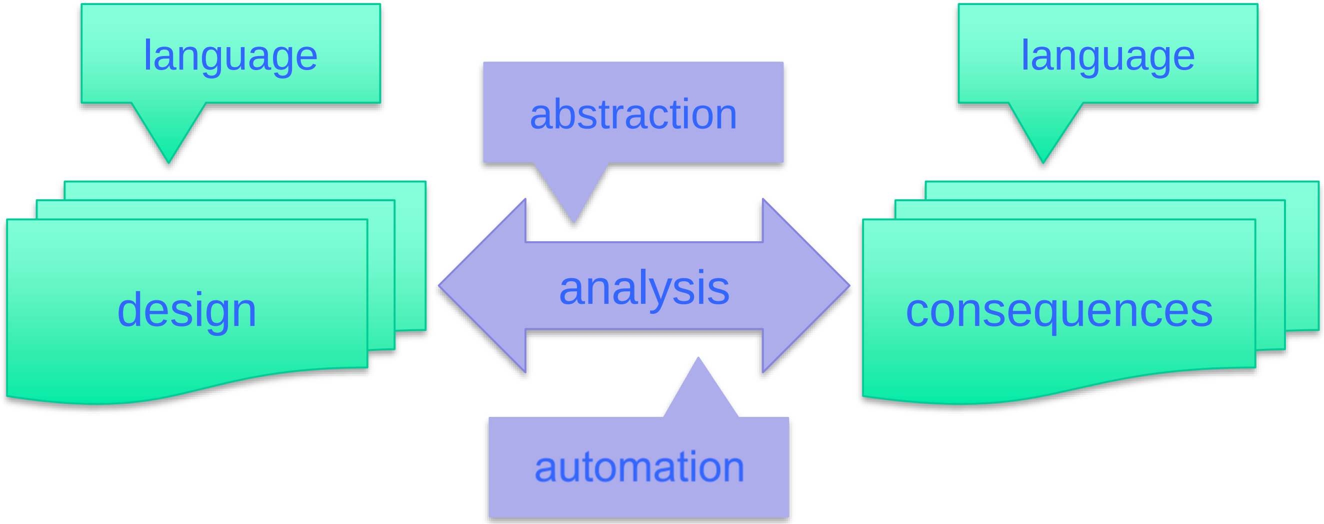
A Few Words About Rigor

- **Rigor in engineering is a distinguishing virtue**
  - it's what we do
- **Rigor requires no justification and we offer none**
- ***Rigorous* does not mean *detailed***
  - it means simplification must be justified
- **Rigor is not a value to be traded against time or money**
- **Rigor applies to all endeavors, simple and complex**
- **Rigor applies to all projects, large and small**
- **Rigor leads to**
  - better understanding of mission objectives and constraints
  - more precise descriptions of design concepts and realizations
  - more thorough and principled verification and validation
  - earlier and more effective remediation of defects
  - more accurate projections of budget and schedule

Where Do We Find Rigor?

- **Rigor in engineering manifests itself in three dimensions**
  - we use precise language to *describe* things
  - we use mathematical abstractions to *analyze* things
  - we use automation for both

- **Putting it together….**



Language

- **We can’t analyze what we can’t describe**
- **We can’t describe precisely without precise language**
- **Mature engineering disciplines define precise descriptive terms and taxonomic relationships**
  - e.g., resistor, capacitor, filter, amplifier, etc.
- **Mature engineering disciplines define composition rules that let us aggregate terms into “sentences” with clear meaning**
  - e.g., SPICE netlist circuit description
- **Precise languages generally manifest the following:**
  - vocabulary: terms
  - syntax: rules for constructing sentences
  - semantics: meaning in the real world
- **Real-world meaning in engineering comes from analysis**

Image credit: wikimedia commons

Abstraction

- **Abstractions are the key to analysis**
- **For example, an RC circuit can be modeled by a linear ordinary differential equation**
  - The equation is an abstraction in that it is a purely mathematical description of *idealized* behavior
  - We can perform operations on this abstraction; in fact we can *solve* it
  - The solution is a useful approximation of the *actual* behavior of the filter
- **Mathematical analysis is a hallmark of engineering**
  - Everything else is poetry or marketing or ….
- **The scope of applicable math has enlarged over time**
  - No longer just calculus, linear algebra and probability
  - Now formal logic, graph theory, abstract algebra, etc.
  - For example, telecom error-correcting codes employ algebra proudly claimed to be useless until the 20th century

Language and Abstraction

- **What is a capacitor?**
  - Is it necessarily a discrete component?
- **A better definition: something that exhibits *capacitance***
- **And what is capacitance?**
  - a specific analytical relationship between voltage and current: $I = C\, dV/dt$
- **Note the fundamental linkage of language to abstraction**
  - *capacitor* if and only if $I = C\, dV/dt$
- **This is true of engineering language in general**
  - What we say has *direct* analytical consequences
- **Abstractions shape language and vice-versa**
  - e.g., Modelica is a language of differential-algebraic equations

Image credit: pixabay.com

Automation

- **Automation is critical for engineering because it preserves rigor: scrupulous adherence to the highest standards for the conduct of work**
  - Machines don't cut corners
- **Automation has its own abstractions (e.g., algorithms, data structures)**
- **These abstractions can be mapped to the abstractions of engineering analysis**
  - transitive closure maps to root cause analysis
- **Automation is fundamental to modern engineering because**
  - Well-designed languages are amenable to machine parsing
  - Many useful mathematical abstractions and related analyses are implemented in software libraries
  - Derivation of consequences of design can be automated
  - Design synthesis can be automated

What About Systems Engineering?

- **Systems engineers describe and analyze, but how well?**
- **Is Systems Engineering rigorous?**
- **Do we use precise language?**
- **Do we employ abstractions to empower analysis?**
- **Do we automate effectively?**
- **How can we do better?**

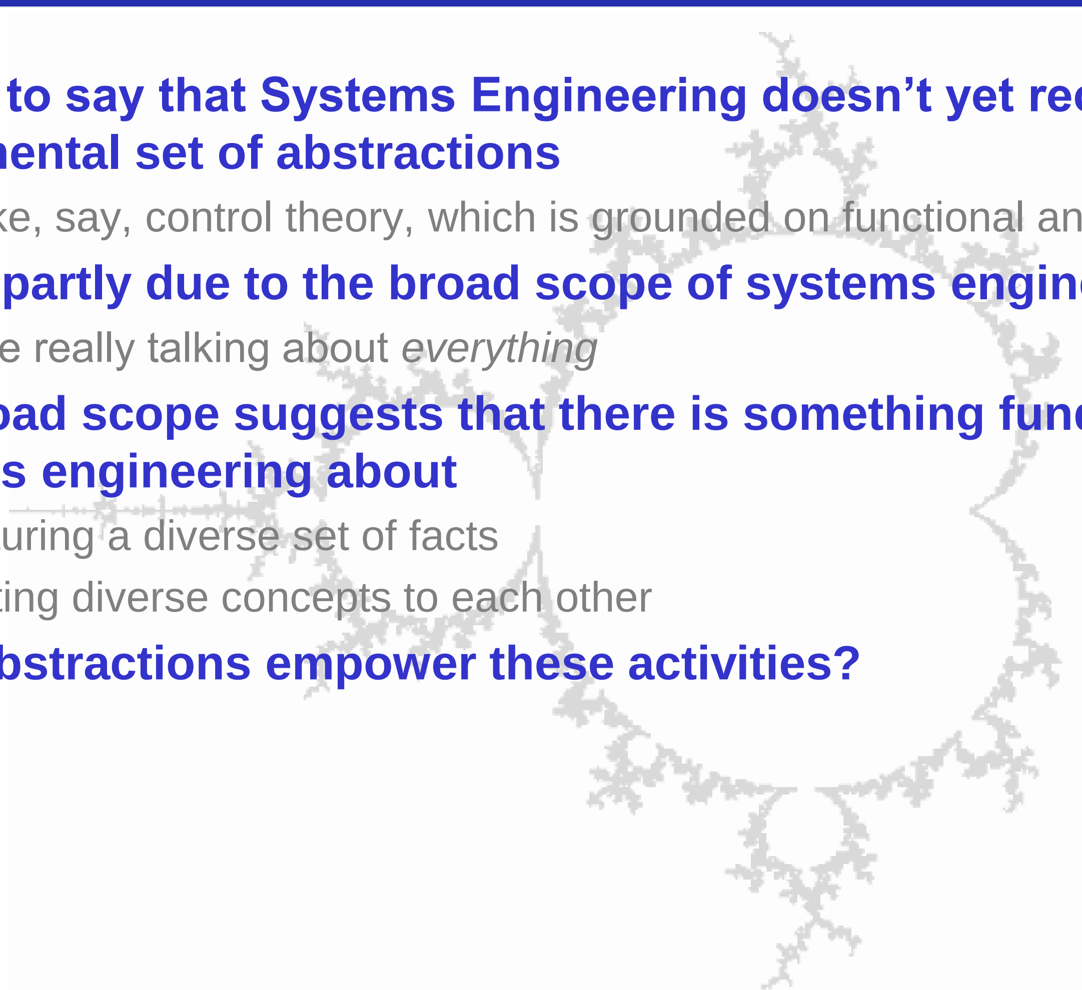
Systems Engineering Language

- **It's fair to say that Systems Engineering employs distinct concepts: *component*, *function*, *interface*, *requirement*, *risk*, etc.**
- **It's also fair to say that we use some words frequently without being very clear about meaning**
  - e.g., *system* vs. *subsystem*
- **As a discipline, we lack agreement on**
  - names for concepts
    - I call it *component*; what do you call it?
  - names for properties
    - How do we refer to an element's name? Its mass?
  - names for relationships
    - What's the relationship between a component and a function?
  - syntax for valid expressions composing concepts, properties, relationships
    - Can a function be performed by more than one component?

How Can We Do Better?

1. **First and foremost, recognize that there is well-established field of theory, practice, and technology dedicated to precise representation of knowledge**
   - called (obviously) *Knowledge Representation*
2. **Use the tools of Knowledge Representation and the Semantic Web to build communities of consensus around systems engineering language usage**
   - captured in formal ontologies
3. **Incorporate this consensus into, not just tools and software, but human language**
   - We should *talk to each other* using our language
4. **Incorporate this consensus into tools and software**
   - Particularly, SysML
5. **Reject ambiguity from our practices**
   - Being precise about uncertainty is good
   - Being ambiguous about anything is not

Systems Engineering Abstractions



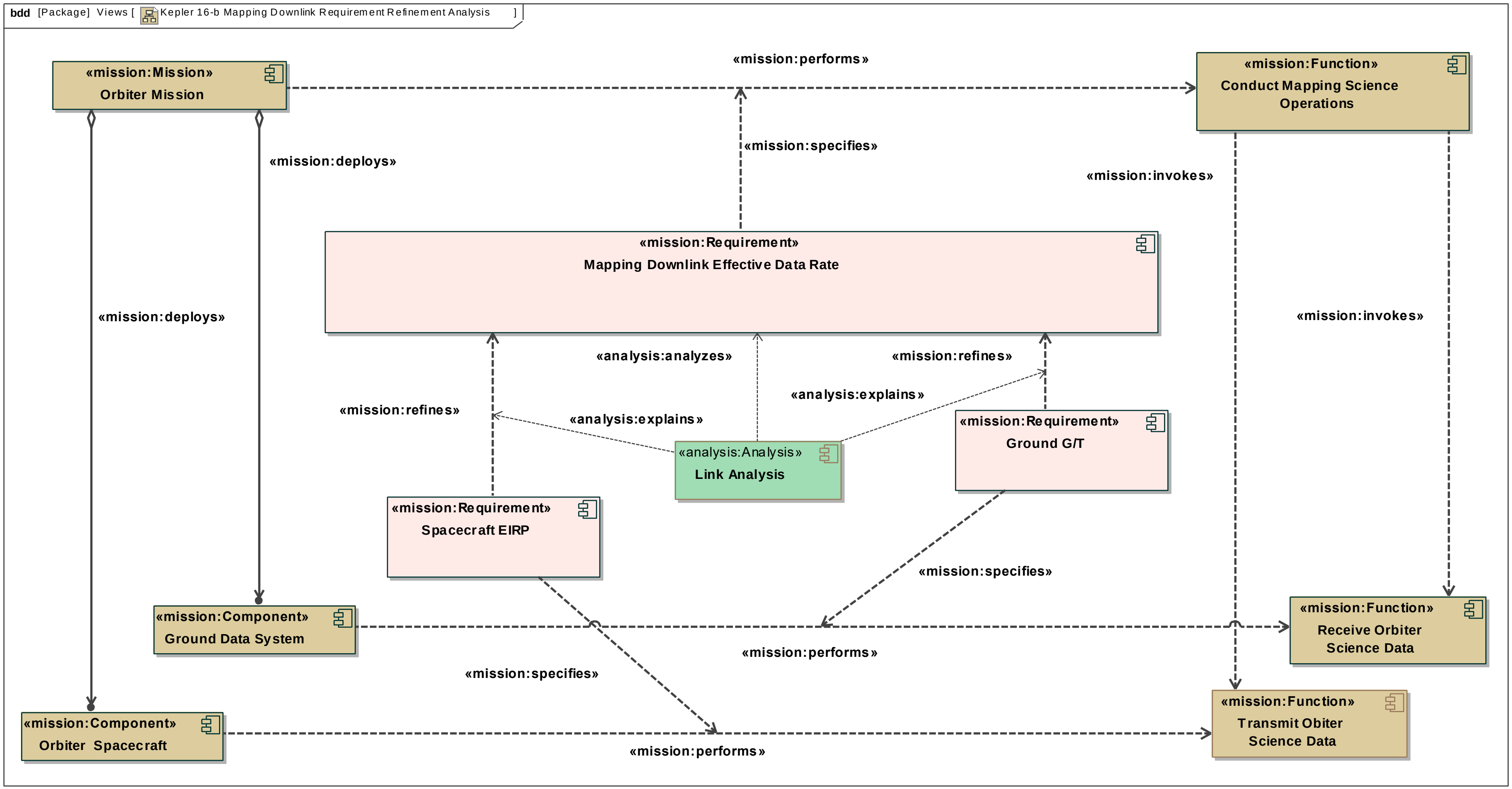
- **It's fair to say that Systems Engineering doesn't yet recognize a fundamental set of abstractions**
  - unlike, say, control theory, which is grounded on functional analysis
- **This is partly due to the broad scope of systems engineering**
  - we're really talking about *everything*
- **The broad scope suggests that there is something fundamental to systems engineering about**
  - capturing a diverse set of facts
  - relating diverse concepts to each other
- **What abstractions empower these activities?**

Image credit: wikimedia commons

How Can We Do Better?

1. **Recognize that graph theory is the mathematical study of graphs, which represent pairwise relations between objects**
2. **Knowledge representation theory makes heavy use of graphs**
3. **We can use graph theory to structure and organize the facts (language assertions) about the objects of our design and analysis**
   - We can reason about whether the resulting graph is well-formed according to the rules of our language
   - We can reason about all kinds and degrees of relatedness
     - e.g., What requirements does this requirement directly refine?
     - Indirectly?
4. **Well-known graph algorithms have direct application**
   - connected components: fault propagation
   - transitive closure: state reachability
   - topological sort: root cause analysis

Example: Knowledge as a Graph



Systems Engineering Automation

- **In the lifetime of the Systems Engineering discipline, computing has gone from a scarce, precious resource to a commodity**
- **Have we, as a discipline, taken advantage of that?**
- **There are all kinds of important analyses that are computationally-intensive**
  - logical reasoning
  - search
  - planning and scheduling
  - feasible region bounding
- **Graph theory is fundamental to computation as well**

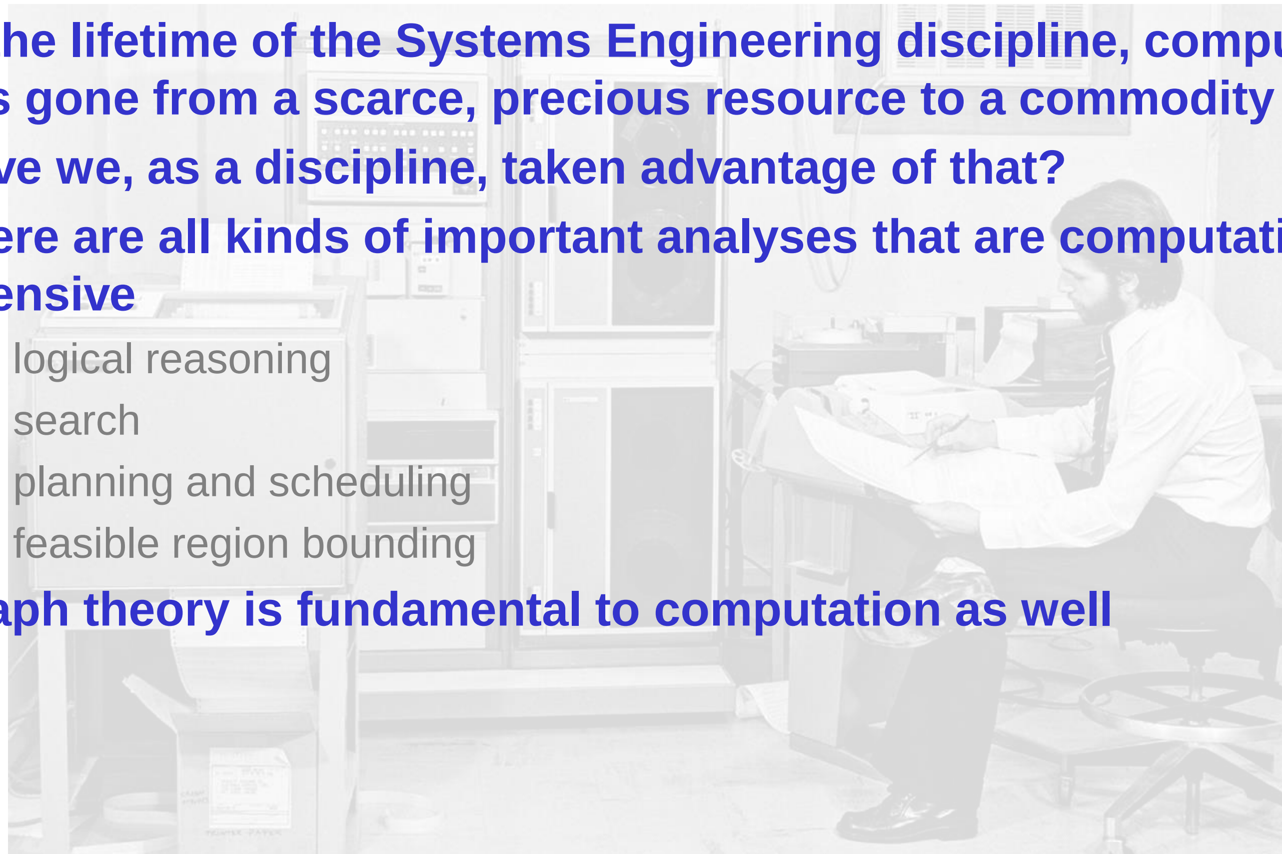


Image credit: wikimedia commons

Conclusion

- **Systems Engineering is really Engineering to the degree that it achieves rigor through in description and analysis through**
  - precise language with rules and meaning
  - mathematical abstractions
  - automation
- **Graph theory is a fundamentally applicable abstraction that empowers both description and analysis**
- **I don't like the term *Model-Based Systems Engineering* because it leads to silly questions like "What is a Model?"**
- **But I would *describe* MBSE as Systems Engineering practice that achieves rigor through use of**
  - **precise language for description**
  - **mathematical abstractions for analysis**
  - **effective automation**

Thank You

Questions?

Steven Jenkins

J.S.Jenkins@jpl.nasa.gov